Build 1: Control an LED with NFC

Description:

This was a build to control an LED with an NFC. The build contains the Arduino RFID/NFC shield, an Arduino, an LED, a 1KΩ resistor, and 4 wires.

Issues:

The biggest issue I ran into was a lack of understanding about how the NFC reader worked. I assumed that the NFC reader would constantly be on and if there wasn't a card in front of it, nfc.readPassiveTargetID would return a zero. But in practice, it didn't do that. I'm not sure if it was just that the reader wouldn't return a value at all, or it wasn't activated until there was a card above it, but the LED wouldn't turn off until I coded it to turn off.

I was surprised it didn't work because most other things we've used haven't read anything but took direct input from the user. I assumed that reading a card would be like direct input, except it was constantly being read. I learned that NFC/RFID readers aren't constantly scanning, otherwise, the value of readPassiveTargetID would change. Instead it seems more like they look for a card and hold onto that value until another card is used.

Diagram of Build

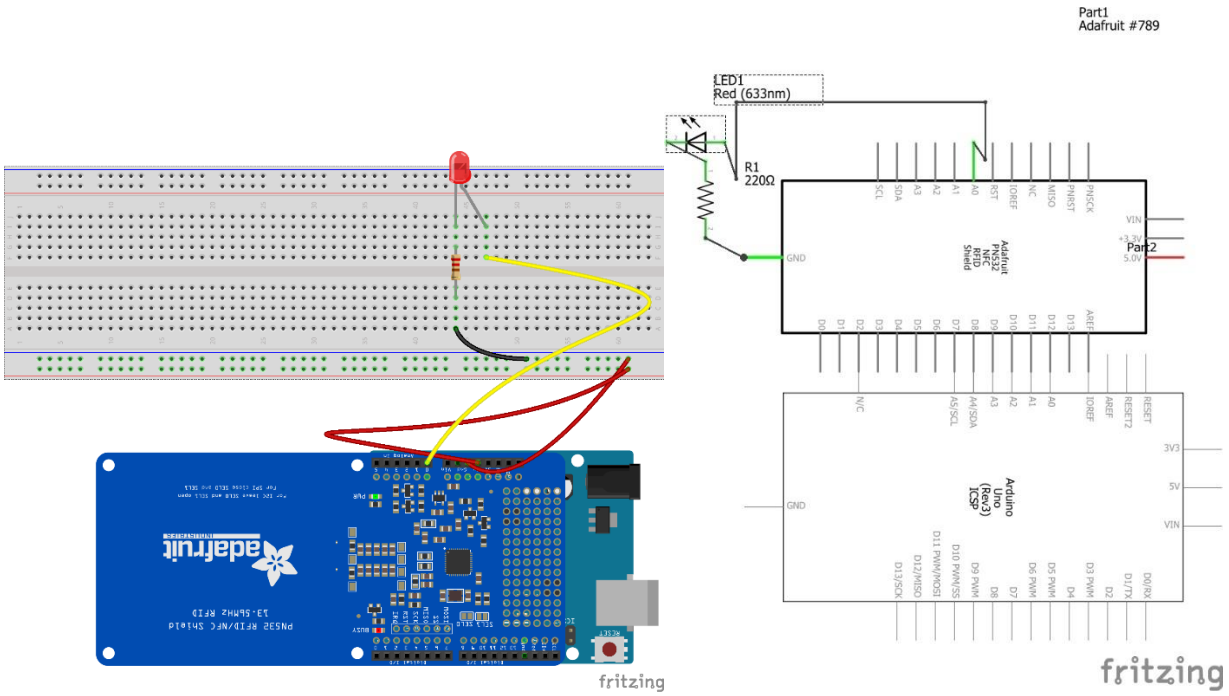


Photo of Build

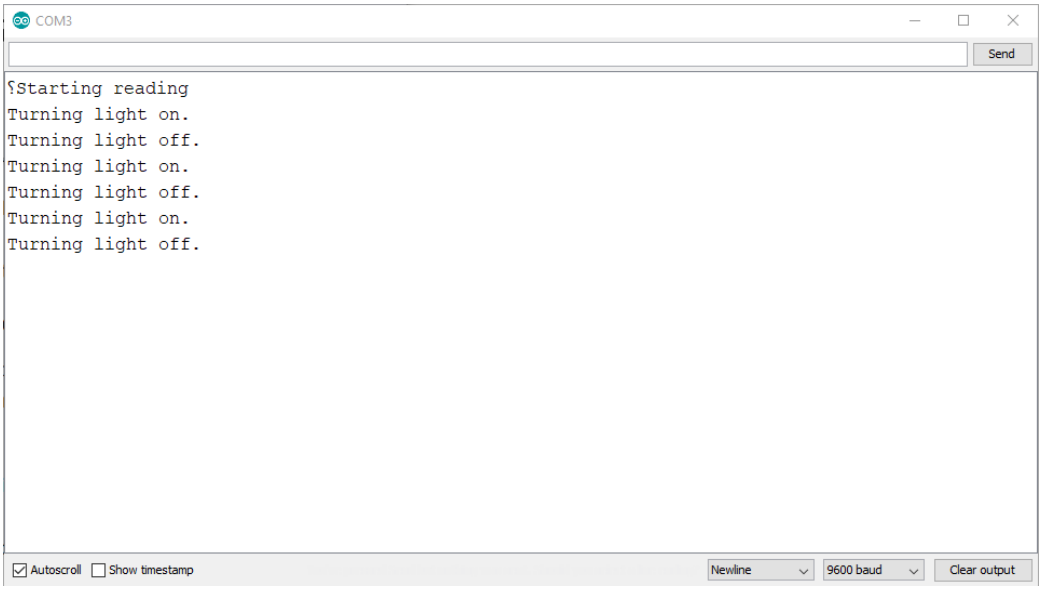
Code for Build

```
/*************************************************************************
*****/

/*! The code for controlling and reading the NFC card is from
readMifare.pde by

 * Adafruit Industries


*/

/*************************************************************************
*****/

#include <Wire.h>

#include <SPI.h>

#include <Adafruit_PN532.h>


// If using the breakout with SPI, define the pins for SPI communication.

#define PN532_SCK  (2)

#define PN532_MOSI (3)

#define PN532_SS   (4)

#define PN532_MISO (5)


#define LED_PIN A0
```

```
// If using the breakout or shield with I2C, define just the pins connected
// to the IRQ and reset lines.  Use the values below (2, 3) for the shield!
#define PN532_IRQ   (2)
#define PN532_RESET (3)  // Not connected by default on the NFC Shield

// Or use this line for a breakout or shield with an I2C connection:
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);

#if defined(ARDUINO_ARCH_SAMD)
   #define Serial SerialUSB
#endif

void setup(void) {
  #ifndef ESP8266
    while (!Serial); // for Leonardo/Micro/Zero
  #endif
  Serial.begin(9600);
  Serial.println("Starting reading");
  pinMode(LED_PIN, OUTPUT);
  nfc.begin();

  nfc.SAMConfig();
}

void loop(void) {
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
```

```
  uint8_t uidLength;                        // Length of the UID (4 or
7 bytes depending on ISO14443A card type)


  success   =   nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A,   uid,
&uidLength);


  if (success) {
//    nfc.PrintHex(uid, uidLength);
    Serial.println("Turning light on.");
    analogWrite(LED_PIN, 255);
    delay(5000);
    Serial.println ("Turning light off.");
    analogWrite(LED_PIN, 0);
  }


  delay(2000);
}
```

Example of Output



Build 2: Doorbell Prototype

Description:

This was a build to control a Piezo buzzer with an NFC, playing different tones based on the NFC's UID. The build contains the Arduino RFID/NFC shield, an Arduino, a Piezo buzzer, and 4 wires.

Issues:

I didn't have any issues with this build, but I did learn a little bit more about how the NFC/RFID shield works. So, in this example, it seemed as if the value written to whichever output pin I use stays constant regardless of whether the initial card I used is there. The only time it changes is when I use a different RFID card. Without some code limiting how long the sent value executes, Arduino will keep using the previous value. It makes me wonder if there is some way to change this behavior in code so that the RFID/NFC reader is always replacing the prior value until a new card is scanned.

The successes of the project were that I ultimately got it to work. I wouldn't call this a failure, but I edited my code so that the tone didn't keep playing until another card was waved over it.

Diagram of Build

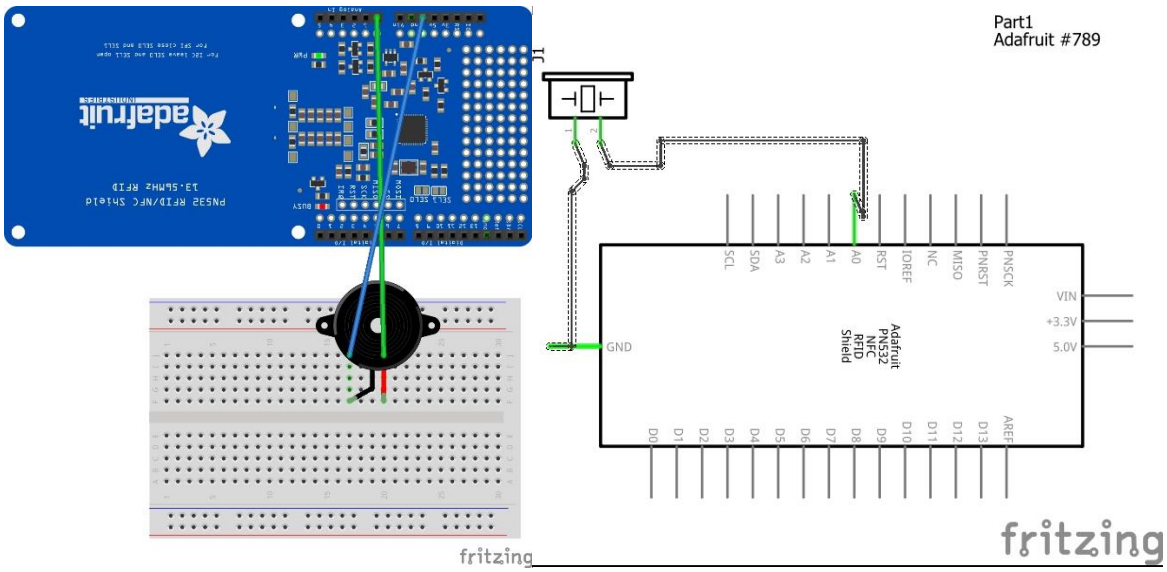
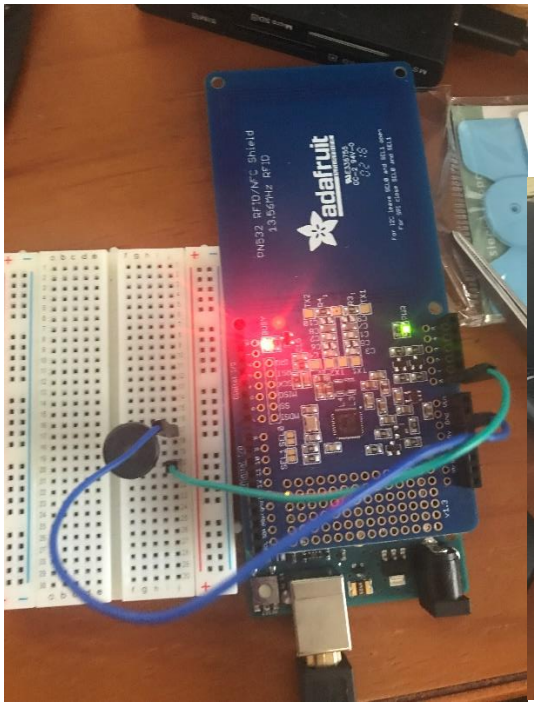
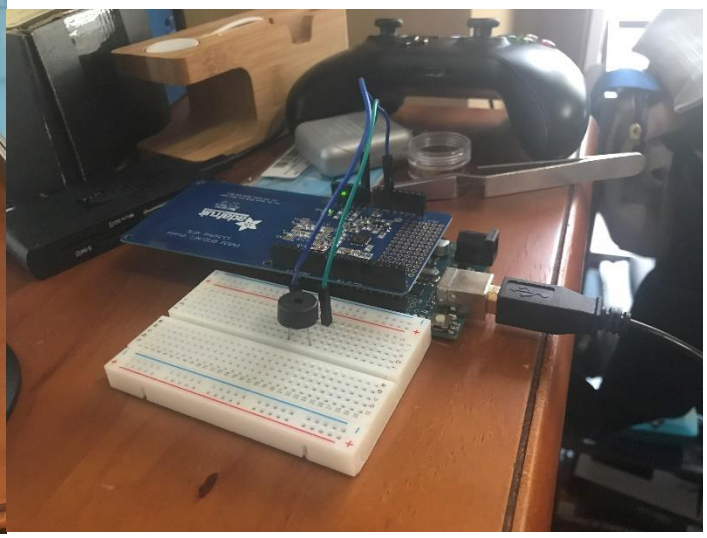


Photo of Build





Code for Build

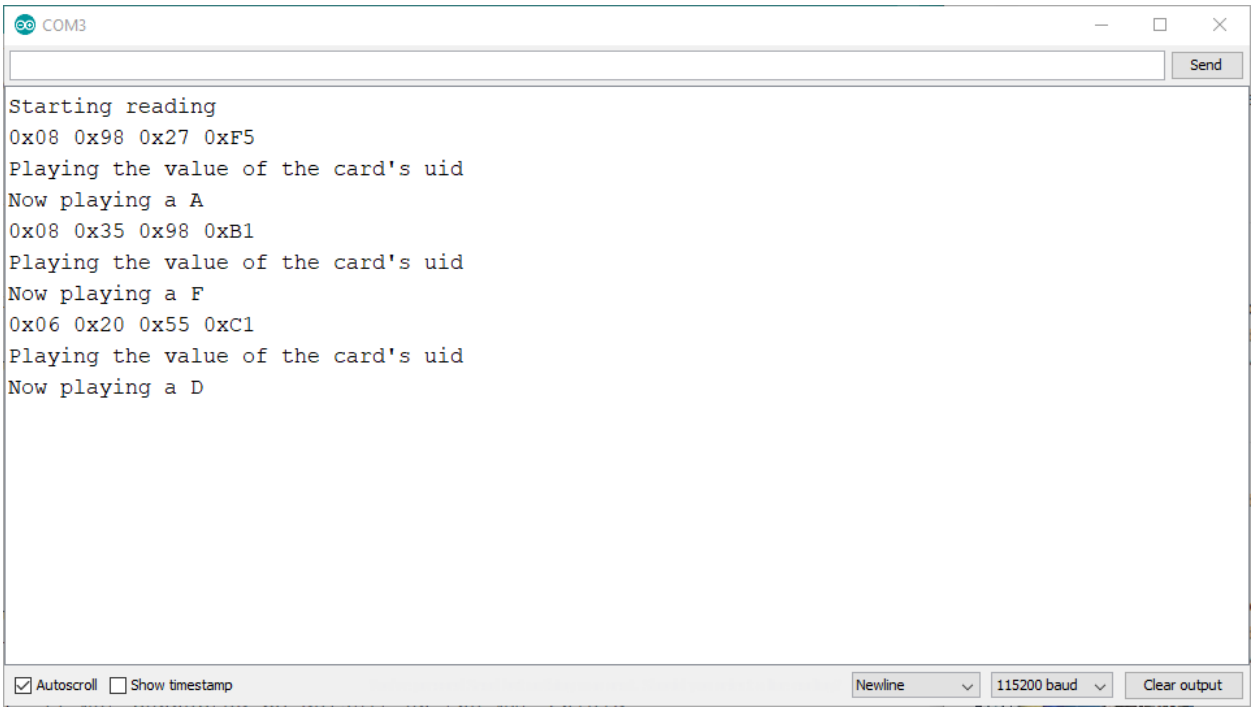
```
/**************************************************************************
*****/

/*! The code for controlling and reading the NFC card is from
readMifare.pde by

 * Adafruit Industries


*/

/**************************************************************************
*****/

#include <Wire.h>

#include <SPI.h>

#include <Adafruit_PN532.h>


// If using the breakout with SPI, define the pins for SPI communication.

#define PN532_SCK  (2)

#define PN532_MOSI (3)

#define PN532_SS   (4)
```

```
#define PN532_MISO (5)


#define SPEAKER_PIN A0


// If using the breakout or shield with I2C, define just the pins connected
// to the IRQ and reset lines.  Use the values below (2, 3) for the shield!
#define PN532_IRQ   (2)
#define PN532_RESET (3)  // Not connected by default on the NFC Shield


// Or use this line for a breakout or shield with an I2C connection:
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);


#if defined(ARDUINO_ARCH_SAMD)
   #define Serial SerialUSB
#endif


void setup(void) {
  #ifndef ESP8266
    while (!Serial); // for Leonardo/Micro/Zero
  #endif
  Serial.begin(115200);
  Serial.println("Starting reading");
  pinMode(SPEAKER_PIN, OUTPUT);
  nfc.begin();


  nfc.SAMConfig();
}
```

```
void loop(void) {
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the
returned UID
  uint8_t uidLength;                        // Length of the UID (4 or
7 bytes depending on ISO14443A card type)
  int value = 0;

  // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
  // 'uid' will be populated with the UID, and uidLength will indicate
  // if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)
  success   =   nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A,   uid,
&uidLength);

  if (success) {
    nfc.PrintHex(uid, uidLength);
    for (int i = 0; i < 7; i++) {
      value += uid[i];
    }

    Serial.println("Playing the value of the card's uid");

    if(value >= 261 && value < 294) {
      Serial.println("Now playing a C");
    }

    if(value >= 294 && value < 329) {
      Serial.println("Now playing a D");
    }

    if(value >= 329 && value < 349) {
```

```
      Serial.println("Now playing an E");
    }

    if(value >= 349 && value < 392) {
      Serial.println("Now playing a F");
    }

    if(value >= 392 && value < 440) {
      Serial.println("Now playing a G");
    }

    if(value >= 440 && value < 493) {
      Serial.println("Now playing an A");
    }

    if(value >= 493 && value < 523) {
      Serial.println("Now playing a B");
    }

    if(value >= 523 || value < 261) {
      Serial.println("Entering a different octave");
    }

    tone(SPEAKER_PIN, value);
    delay(10000);
    noTone(SPEAKER_PIN);
  }
  delay(2000);
}
```

Example of Output


COM3
Send
Starting reading
0x08 0x98 0x27 0xF5
Playing the value of the card's uid
Now playing a A
0x08 0x35 0x98 0xB1
Playing the value of the card's uid
Now playing a F
0x06 0x20 0x55 0xC1
Playing the value of the card's uid
Now playing a D
Autoscroll
Show timestamp
Newline
115200 baud
Clear output

Build 3: NFC Lock Prototype

Description:

This was a build to control a motor based on whether the correct NFC card is used. This build contains an H-bridge, a stepper motor, an NFC/RFID reader for the arduino, and multiple wires.

Issues:

For this build, the biggest issue was figuring out how to read the UID of the NFC card. None of the functions in the Adafruit library for the NFC/RFID reader allowed you to just read and store the UID for the cards being used. They just printed them. I ultimately had to write a function just to store the value, but I kept running into issues when comparing the UID value with the key value that I saved. It turned out the issue was I was assuming arduino had a way to compare complete array against each other when it didn't. Once I split compared each element of the UID array against the key value, my program worked as planned.

I think I learned more about how the Arduino IDE works from this build. I was surprised by how limited the functionality was. I think most of this semester, I assumed that Arduino was closer to Java regarding having built in functions and logic that would do things for you with some C elements. I also learned a little more about the logic that goes into smart locks, or at least keycode locks. One thing I want to work on in the future is trying to figure out how to sort a key value initially rather than have a preset key value.

Diagram of Build

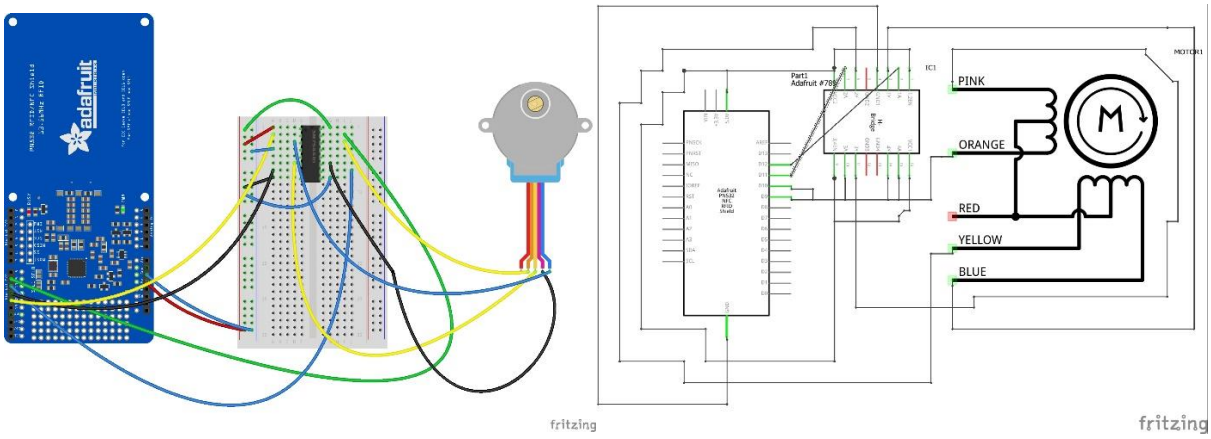
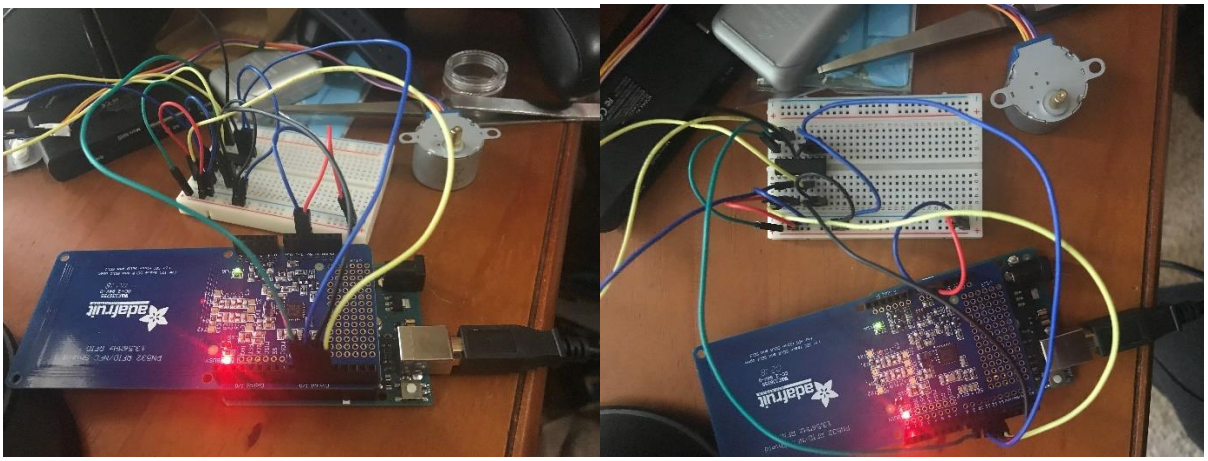


Photo of Build



Code for Build

```
/*************************************************************************
*****/
/*! The code for controlling and reading the NFC card is from
readMifare.pde by
 * Adafruit Industries. Motor code from
 *      https://learn.adafruit.com/adafruit-arduino-lesson-15-dc-motor-
reversing/arduino-code.
*/
/*************************************************************************
*****/
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_PN532.h>
#include <Stepper.h>

// If using the breakout with SPI, define the pins for SPI communication.
#define PN532_SCK  (2)
#define PN532_MOSI (3)
#define PN532_SS   (4)
#define PN532_MISO (5)

#define SPEAKER_PIN A0

// If using the breakout or shield with I2C, define just the pins
connected
// to the IRQ and reset lines.  Use the values below (2, 3) for the
shield!
#define PN532_IRQ   (2)
#define PN532_RESET (3)  // Not connected by default on the NFC Shield

// Or use this line for a breakout or shield with an I2C connection:
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);

#if defined(ARDUINO_ARCH_SAMD)
```

```
   #define Serial SerialUSB
#endif

const int stepsPerRevolution = 512; //number of steps per revolution for
the motor

const uint8_t key[4] = {6, 32, 85, 193};


int in1Pin = 12;
int in2Pin = 11;
int in3Pin = 10;
int in4Pin = 9;

int speed = 30;
int steps = 0;
boolean state = true;
boolean open = false;

Stepper motor(stepsPerRevolution, in1Pin, in2Pin, in3Pin, in4Pin);

void setup(void) {
  #ifndef ESP8266
    while (!Serial); // for Leonardo/Micro/Zero
  #endif
  Serial.begin(115200);
  Serial.println("Starting reading");
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  pinMode(in3Pin, OUTPUT);
  pinMode(in4Pin, OUTPUT);
  nfc.begin();

  nfc.SAMConfig();

//  uint8_t success;
//  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the
returned UID
//  uint8_t uidLength;                        // Length of the UID (4
or 7 bytes depending on ISO14443A card type)
//  int value = 0;
//
//    success  =  nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A,  uid,
&uidLength);
//  if(success) {
//    Serial.println("Saving key");
//    save_value(uid, uidLength, key);
//    for(int i = 0; i < 7; i++) {
//      Serial.print(key[i]);
//      Serial.print(" ");
//    }
//
```

```
//    Serial.println("");
//  }
}


void loop(void) {
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the
returned UID
  uint8_t uidLength;                        // Length of the UID (4 or
7 bytes depending on ISO14443A card type)
  int value = 0;

  success   =   nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A,   uid,
&uidLength);

  if (success) {
    uint8_t holder[4];
    save_value(uid, uidLength, holder);

    for(int i = 0; i < uidLength; i++) {
      if (holder[i] != key[i]) {
        state = false;
      }
    }

    if(state) {
      Serial.println("Welcome home.");
      motor.setSpeed(speed);
      Serial.println("Unlocking door. You have 10 seconds to enter.");
      motor.step(512);
      open = true;
      delay(10000);
      Serial.println("Locking door.");
      motor.step(-512);
      open = false;
    }

    else {
      Serial.println("Unregistered key. Step away from the door or 911
will be called.");
      if (open) {
        motor.step(-512);
      }
    }
  }

  delay(2000);
}
```

```
void save_value(const byte * data, const uint32_t numBytes, uint8_t
*holder)
{
  uint32_t szPos;
  for (szPos=0; szPos < numBytes; szPos++)
  {
    holder[szPos] = data[szPos];
    Serial.println(data[szPos]);
  }
}
```

Example of Output